IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants	:	Martin Thomas MILLER et al.
Serial No.	:	09/988,416
For	:	PROCESSING WEB FOR DATA PROCESSING IN A DIGITAL OSCILLOSCOPE OR SIMILAR INSTRUMENT
Filed	:	November 16, 2001
Examiner	:	Jeffrey R. West
Art Unit	:	2857

DO NOT ENTER -JRW

745 Fifth Avenue
New York, NY 10151

CERTIFICATE OF ELECTRONIC FILING

I hereby certify that this correspondence is being transmitted via Electronic Filing Services on December 13, 2007.

Susan Powell
(Name of person signing transmittal)

Susan Powell
Signature

December 13, 2007
Date of Signature

AMENDMENT UNDER 37 CFR 1.116

Mail Stop **AF**
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This is in response to the Final Office Action dated September 11, 2007. Please extend the term of response to this Office Action by one month to January 11, 2007. Payment of the fee for this extension of time is submitted electronically herewith.

Please amend the above-identified application as follows:

LabVIEW Summary and Review

Part I. LabVIEW Basics

Objectives:

Introduction to National Instrument's LabVIEW Graphical Computing Language.
Numeric, Boolean and Character Controls and Indicators used in LabVIEW.
For Loop, While Loop, Case, Sequence and Formula Node Structures used in LabVIEW.

Reference:

Lisa K. Wells and Jeffrey Travis, LabVIEW for Everyone: Graphical Programming Made Even Easier, Prentice Hall, 1997
Robert H. Bishop, Learning with LabVIEW 6i, 2nd Edition, Addison-Wesley, 2001.
National Instruments, LabVIEW Support on the web: http://www.ni.com/support/lvsupp.htm.

Introduction:

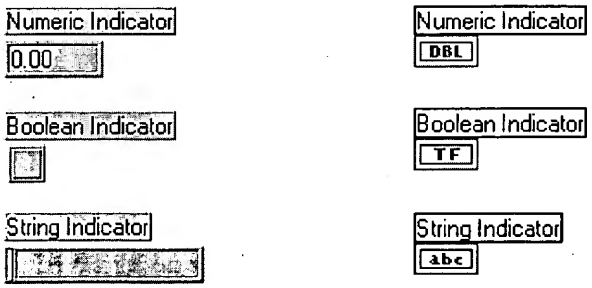
LabVIEW is a program development application, much like C or FORTRAN. LabVIEW is, however, different from those applications in one important respect. Other programming systems use text-based languages to create lines of code, while LabVIEW uses a graphical programming language, G, to create programs in block diagram form.

LabVIEW, like C or FORTRAN, is a general-purpose programming system with extensive libraries of functions for many programming tasks. LabVIEW includes libraries for data acquisition, data analysis, data presentation, and data storage. A LabVIEW program is called a virtual instrument (VI) because it's appearance and operation can imitate an actual instrument.

A VI consists of two panels; one is the **front panel,** and the other is the **block diagram.** These are defined below:

The interactive user interface of a VI is called the **front panel**, because it simulates the panel of a physical instrument. The front panel can contain knobs, push buttons, graphs, and other controls and indicators. You enter data using a mouse and keyboard, and then view the results on the computer screen.

The VI receives instructions from a **block diagram**, which you construct in G. The block diagram is a pictorial solution to a programming problem. The block diagram is also the source code for the VI.



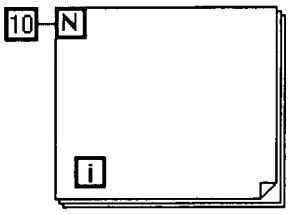
Indicators:

Indicators are used to output numeric (integer or floating point), character, and Boolean data in LabVIEW. On the block diagram, indicators are represented with a thin border.

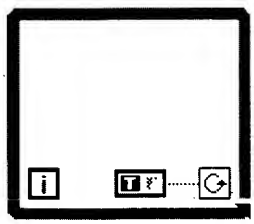
Controls:

Controls are used to input numeric (integer or floating point), character, and Boolean data in LabVIEW. On the block diagram, controls are represented with a thick border.

For Loop Structure:



A For Loop executes its subdiagram N times, where the count equals the value contained in the count terminal. You set the count explicitly by wiring a value from outside the loop to the left of the count terminal. The iteration terminal, i, contains the current number of completed iterations; 0 during the first iteration, 1 during the second, and so on up to N-1. If you wire 0 to the count terminal, the loop does not execute.

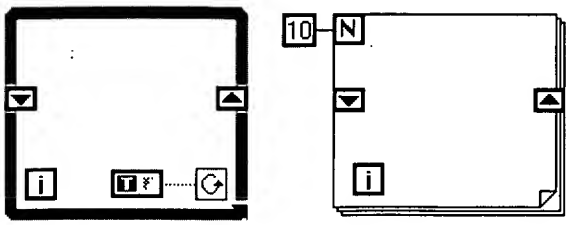


While Loop Structure:

A While Loop executes its subdiagram until a Boolean value you wire to the conditional terminal is FALSE. LabVIEW checks the conditional terminal value at the end of each iteration, and if the value is TRUE, another iteration occurs, so the loop always executes at least once. The default value of the conditional terminal is FALSE, so if it is unwired, the loop iterates only once. The iteration terminal behaves exactly as it does in the For Loop. In LabVIEW 6i, there is also a "stop" termination for the while loop; i.e., the loop will continue to execute until

the stop condition is true.

Shift Registers:

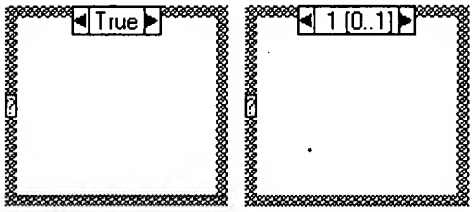


Both loop structures can have terminals called shift registers that you use for passing data from the current iteration to the next iteration. Shift registers are local variables that feed forward or transfer values from the completion of one iteration to the beginning of the next.

A shift register has a pair of terminals directly opposite each other on the vertical sides of the loop border. The right terminal, the rectangle with the up arrow, stores the data at the completion of the iteration. LabVIEW shifts that data at the end of the iteration, and it appears in the left terminal, the rectangle with the down arrow, in time for the next iteration. You can use shift registers for any type of data, but the data you wire to each register terminals must be of the same type.

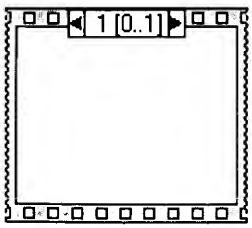
To initialize a shift register, wire a value from outside the loop to the left terminal. If you do not initialize the register, the loop uses as the initial value the last value inserted in the register when the loop last executed, or the default value for its data type if the loop has never before executed. You should normally use initialized shift registers to ensure consistent behavior.

Case Structure:



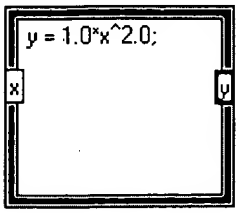
The Case Structure has two or more subdiagrams, or cases, of which only one will execute when the structure executes. This depends on the value of the Boolean or numeric scalar you wire to the external side of the selection terminal. If a Boolean is wired to the selector, the structure must have two cases, False and True. If a numeric is wired to the selector, the structure can have from 0 to n cases.

Sequence Structure:



The Sequence Structure, which looks like a frame of film, consists of one or more subdiagrams, or frames, that execute sequentially. Determining the execution order of a program by arranging its elements in sequence is called control flow. A Sequence Structure executes Frame 0, followed by Frame 1, then Frame 2, until the last frame executes. Only when the last frame completes does data leave the structure.

Formula Node Structure:



The Formula Node contains one or more formula statements delimited by a semicolon. Formula statements use syntax similar to most text-based programming languages for arithmetic expressions. The pop-up menu on the border of the Formula Node contains options to add input and output variables. Math functions must be lowercase.

A thicker border distinguishes the output variables. There is no limit to the number of variables or formulas in a Formula Node. All inputs and outputs must be denoted on the border, even if they are not used outside the structure.

Part II. Arrays and Graphs in LabVIEW

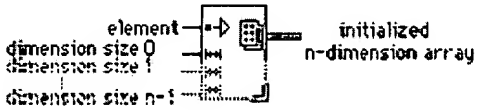
Objectives:

Introduction to 1D and 2D arrays used in LabVIEW.
Introduction to XY Graph used in LabVIEW.

Introduction:

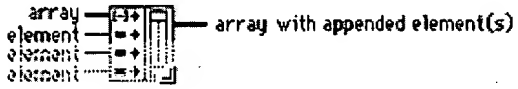
LabVIEW provides a number of functions for working with 1D and 2D arrays. For MEL 2 applications, we only need to work with three array functions. Those functions are Initialize Array, Build Array and Index Array. After learning how to use these array functions, we will then learn how to use them when making an XY Graph.

Initialize Array



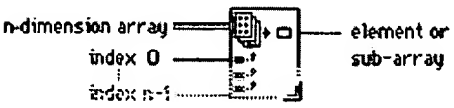
Returns an n-dimensional array in which every element is initialized to the specified value. This function is resizable, so we typically define an array of one element. The element cannot be an array.

Build Array



Concatenates inputs in a top-to-bottom order. Pop-up on an input node and select Change to Array to change it to an array input. For an n-dimensional array, element inputs must have n -1 dimensions, and array inputs must have n dimensions.

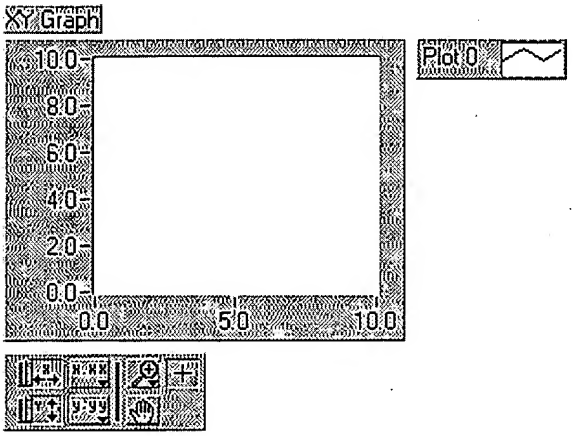
Index Array



Returns an element of the array at the index input. If the array is multi-dimensional you must add additional

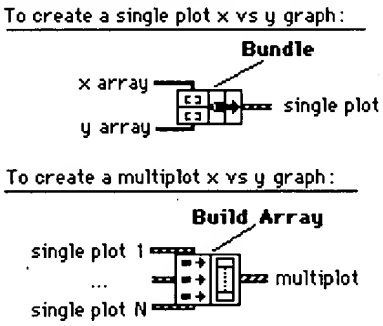
index terminals by resizing or popping up and adding terminals. You can slice out sub-arrays (e.g. rows or columns) by disabling the index terminals from the popup.

XY Graph



A graph indicator is a two-dimensional display of one or more plots. The graph receives and plots data as a block. The XY graph is a general-purpose, Cartesian graphing object that you can use to plot multi-valued functions.

Use this arrangement to bundle two 1D arrays into a cluster, to be plotted.



Use this arrangement to build an array from several clusters.

The input into an XY graph indicator for a single plot is a cluster containing a x array and a y array. You can also display multiple plots on a XY graph by grouping the cluster outputs into a build array function. The above diagrams illustrate how you would create either a single or multiple plots on an XY graph.

Part III. Writing to a Spreadsheet File and Analog Input VI's

Objectives:

Introduction to Writing to a Spreadsheet File VI used in LabVIEW.
Introduction to Analog Input VI used in LabVIEW.

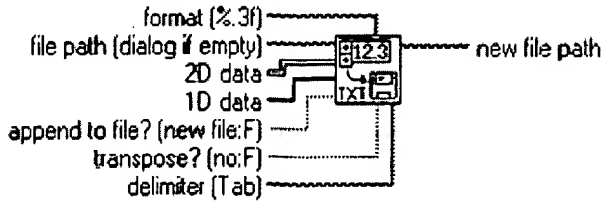
Reference:

Amplifiers: Table 3-2. Actual Range and Measurement Precision (AT-MIO/AI E Series User Manual, see the first page of this manual's appendices).
Table 3-3. MIO I/O Connectors Signal Summary (SC-2043-SG User Manual).

Introduction:

Two VI functions are covered during this laboratory; the Write to Spreadsheet File provides us with a method of storing data values acquired during data acquisition, and Analog Input allows us to take on reading of the analog

input to specified channel on the DAQ board.



Write to a Spreadsheet File

The Write to a Spreadsheet File converts a 2D or 1D array of single-precision numbers to a text string and writes the string to a new file or appends the string to an existing file. This VI opens or creates the file beforehand and closes it afterwards.

File path – the path name of the file. If file path is empty (default value) or is Not a Path, the VI displays a File dialog box from which you can select a file.

2D data – contains the single-precision numbers the VI writes to the file, if 1D data is not wired or is empty.

1D data – contains the single-precision numbers the VI writes to the file if this input is not empty.

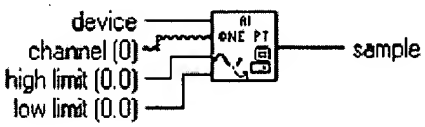
Append to a file (Boolean) – set to TRUE if you want to append the data to existing file. Set to False if you want the data to a new file or to replace an existing file.

Transpose (Boolean) – set TRUE to transpose the data before converting it to a string. The default value is FALSE.

Format (string) – specifies how to convert the numbers to characters.

New file path – is the path of the file to which the VI wrote the data.

AI Sample Channel



Measures the voltage signal attached to the specified channel and returns the measurement value. Use AI Sample Channel to do an immediate measurement of one analog input channel.

Device (integer): the number (typically 1) of the data acquisition board installed in the computer.

Channel (string): specifies the analog input channel to sample.

High limit (single-precision): specifies the maximum-scaled data you expect to measure at the input channel. Use the values listed on the input range column of Table 3-2 in the AT-MIO-AI E Series User Manual.

Low limit (single-precision): specifies the minimum-scaled data you expect to measure at the input channel. Use the only the values listed on the actual input range column of Table 3-2 in the AT-MIO-AI E Series User Manual.

Sample (single-precision): scaled analog input data for the input channel in scaled data units.

Lab Exercises:

You can use the examples located at the directory indicated by the instructor. Also don't forget that you have the online reference material and the LabVIEW book.

1. Create a VI that continuously averages five integer inputs and displays their average, the maximum, and minimum values. Display the average of the inputs as a real (floating point) number with 4 digits of precision. Use a while loop to make the program run continuously, rather than the continuous run button.

2. Create a VI that contains a Sub-VI, where the Sub-VI computes strain. This Sub-VI has three input variables (GF, the gage factor, V_{excite}, the excitation voltage, and V_{out}, the output voltage) and one output variable, ε, the strain. The Sub-VI uses the following equation,

$$\varepsilon = \frac{1}{GF}\frac{4V_{out}}{V_{excite}}$$

Create a VI that collects ten samples of two random numbers scaled to the range 0-5, places these samples in a 10 X 2 array, and then displays only one column of this 2-D array, specified by the user. Display the 10 X 2 array as it is being created and the column of numbers after the collection is completed.

Create a VI that records and plots random numbers once a second. Using a Chart, plot the random numbers as they are generated.

Create a VI that collects samples of potentiometer voltage readings at a user specified time interval. On the front panel, create controls for the channel number and the high and low input voltage limits, and an indicator to display the voltage (sample) using a meter or a gauge. Wire the potentiometer with an excitation voltage of 10 volts, the ground into AIGROUND and the signal voltage (from the swiper) wired into one of the analog input channels 9 through 15 {ACH<9...15>} (refer to Table 3-3 of the SC-2043-SG User Manual).

Modify the VI created in the previous exercise with the ability to continuously store the iteration number and the potentiometer voltage reading with a data format of 5 digits after the decimal. The data file created should be a new data file each time the VI is executed.

Bonus ---

Create a VI that is an Etch-a-Sketch with two potentiometer voltage inputs into the DAQ. Display an XY Graph on the front panel and use the array methods that you learned from the previous exercises.

Note - Store your VI programs on floppy and in your home directory on Rocky.